



Defence Research and
Development Canada

Recherche et développement
pour la défense Canada



Sonar 3 VMSA Federate

User Guide and Technical Description

Allan Gillis

Defence R&D Canada – Atlantic

Technical Memorandum
DRDC Atlantic TM 2005-286
April 2007

Canada

This page intentionally left blank.

Sonar 3 VMSA Federate

User Guide and Technical Description

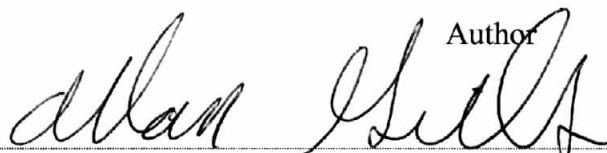
Allan Gillis

Defence R&D Canada – Atlantic

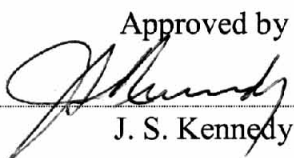
Technical Memorandum

DRDC Atlantic TM 2005-286

April 2007

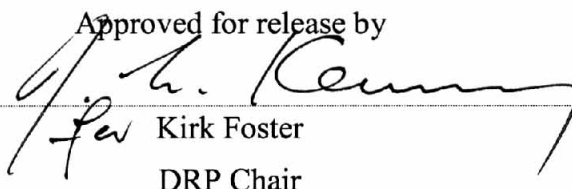
Author


Allan Gillis

Approved by


J. S. Kennedy

Head, Maritime Information and Combat Systems

Approved for release by


Kirk Foster

DRP Chair

Abstract

This federate provides a medium fidelity sonar model for Virtual Maritime Systems Architecture (VMSA) high level architecture (HLA) simulations. The model best represents towed array systems, but can be configured to model flank arrays as well. The Sonar model is coupled with an auto-detector/signal follower that creates VMSA Sonar tracks without an operator.

While the federate is written in Java, the signal follower is provided only as a compiled Windows DLL. This limits the federate to the Windows operating system, either Windows 2000 or XP.

This document describes the federate software, how to use it, and how to integrate the federate into DRDC Atlantic's VMSA execution system. As well, the software design and technical details are explained.

This document covers version 3.0.1 of the Sonar federate.

Résumé

La composante visée ici (« le fédéré ») constitue un modèle sonar de fidélité moyenne pour les simulations d'architecture HLA VSMA (architecture virtuelle des systèmes maritimes). Le meilleur modèle représente des systèmes de réseaux remorqués, mais il peut être configuré pour modéliser également des réseaux de flank. Le modèle sonar est couplé à un autodétecteur/suiveur de signal qui crée des pistes sonar VMSA, sans opérateur.

Bien que le fédéré soit écrit en Java, le suiveur de signal est fourni seulement en DLL Windows compilé, ce qui limite le fédéré au système d'exploitation Windows, soit Windows 2000 ou XP.

Le présent document décrit le fédéré, son utilisation et la façon de l'intégrer au système d'exécution VSMA de RDDC Atlantique. En outre, il explique la conception du fédéré et les détails techniques qui s'y rapportent.

Le présent document couvre la version 3.0.1 de la composante logicielle.

This page intentionally left blank.

Executive summary

Report title

Gillis, A. D.; 2006; Sonar 3 VMSA Federate: User guide and technical description; DRDC Atlantic TM 2005-286; Defence R&D Canada - Atlantic; Unclassified.

Background

The Virtual Maritime Systems Architecture (VMSA) is a framework for distributed simulations in the maritime environment, based on the High Level Architecture (HLA). VMSA was originally developed by the Australian Defence Science and Technology Organisation (DSTO) and is now in use by all of The Technical Co-operation Program (TTCP) countries including Canada.

From the Canadian perspective, sonar is a key federate in almost any simulation of the maritime domain. This version of the sonar federate represents the second sonar federate contributed by Canada to TTCP. The first, Sonar 2.0, was a relatively low fidelity model based on the sonar equations and truth data. It produced bearings only tracks already classified as to surface or subsurface. While Sonar 2 was adequate for many simulation tasks, there was room for considerable improvement.

Sonar 3 makes several small improvements to the underlying simulation for Sonar 2, and introduces a major change in the way tracks are produced. These changes also make Sonar 3 a much better simulation for flank and towed arrays.

Principal results

Sonar 3 significantly improves the fidelity of the Sonar 2 simulation by treating the signatures of targets in the ocean environment as signals that arrive at a sonar array. The signature data of each possible target is added to the beam map of the array, which is then populated with noise data.

The simulated beam map is passed to an external dynamic link library (DLL) that analyses the data, makes detections, and creates tracks (signal followers). Any DLL that implements the same interface can now be used to handle detections and tracks, which makes Sonar 3 a potential platform for testing new algorithms within a simulation environment.

The model used for Sonar 3 best represents towed and flank array systems.

Significance of results

Sonar 3 provides a good fidelity sonar model for sonar arrays within the VMSA. The separation of the detection and tracking code into a separate DLL provides a flexible system for testing detection and tracking algorithms in a simulated environment.

Future work

It is intended that the next sonar federate will include several enhancements to the sonar modelling, user interface, and the software architecture. In particular:

- Shipping noise and surface noise tables to be loaded from text files,
- Handle the characterisation of passive signatures more consistently,
- Include a graphical user interface (GUI) to show the data generated for the beam maps and allow an operator to manually mark and drop tracks,
- Include at least a basic active sonar capability,
- Allow multiple systems to be modelled by a single instance of the federate,
- Move many of the generally useful classes from this federate into the existing VCS utility class library for re-use in other projects.

Sommaire

Titre de rapport

Gillis, A. D.; 2006; Sonar 3 VMSA Federate: User guide and technical description [Fédéré VSMA Sonar 3 : guide d'utilisation et description technique]; RDDC Atlantique TM 2005-286; R & D pour la défense Canada - Atlantique; non classifié.

Situation générale

L'architecture virtuelle des systèmes maritimes (VMSA) offre un cadre pour des simulations distribuées en milieu maritime, basé sur l'architecture HLA. Initialement développée par la Defence Science and Technology Organisation (DSTO) d'Australie, la VMSA est maintenant utilisée par tous les pays du Technical Co-operation Program (TTCP), y compris le Canada.

Du point de vue canadien, le fédéré sonar est un élément clé de presque toutes les simulations du domaine maritime. La présente version est le deuxième fédéré sonar présenté par le Canada au TTCP. Le premier, Sonar 2.0, était un modèle de fidélité relativement faible basé sur les équations sonar et les données de terrain. Il produisait des pistes de relèvement seul déjà classifiées du point de vue surface/sous-marin. Sonar 2 s'est avéré insatisfaisant pour de nombreuses tâches de simulation, mais il se prêtait à d'importantes améliorations.

Sonar 3 apporte plusieurs légères améliorations en matière de simulation et modifie de manière importante la façon dont les pistes sont produites. Ces modifications amélioreront aussi grandement la simulation dans le cas des réseaux de flanc ou remorqués.

Résultats

Sonar 3 améliore sensiblement la fidélité de simulation de Sonar 2 en traitant les signatures des cibles en milieu océanique comme des signaux arrivant à un réseau sonar. Les données de signature de chaque cible possible sont portées sur la carte de faisceaux du réseau, à laquelle sont ensuite ajoutées les données de bruit.

La carte de faisceaux simulée est transmise à une bibliothèque de liens dynamiques (DLL) externe qui analyse les données, effectue les détections et crée des pistes (suiveurs de signaux). Toute DLL qui met en oeuvre la même interface peut maintenant être utilisée pour les détections et les pistes, ce qui fait de Sonar 3 une plate-forme possible pour l'essai de nouveaux algorithmes dans un contexte de simulation.

Le modèle utilisé pour Sonar 3 représente le mieux les systèmes de réseaux de flanc et remorqués.

Portée

Sonar 3 offre un modèle sonar de fidélité satisfaisante pour des réseaux sonar dans le cadre de la VMSA. L'utilisation d'une DLL distincte pour la séparation du code de détection et de pistage donne au système la souplesse nécessaire pour tester des algorithmes de détection et de pistage dans un environnement simulé.

Recherches futures

Le prochain fédéré sonar comprendra plusieurs perfectionnements touchant la modélisation sonar, l'interface utilisateur et l'architecture logicielle, notamment :

- Tableaux des bruits de la navigation maritime et des bruits de surface, à charger à partir de fichiers textes,
- Caractérisation plus uniforme des signatures passives,
- Inclusion d'une interface utilisateur graphique (GUI) pour indiquer les données produites pour les cartes de faisceaux et permettre à un opérateur de marquer et de supprimer manuellement des pistes,
- Inclusion d'au moins une capacité de sonar actif de base,
- Possibilité de modéliser de multiples systèmes au moyen d'une seule instance du fédéré,
- Transfert d'un grand nombre de classes généralement utiles du fédéré à la bibliothèque de classes de l'utilitaire VCS pour réutilisation dans d'autres projets.

Table of contents

Abstract	i
Résumé	i
Executive summary	iii
Sommaire.....	v
Table of contents	vii
List of figures	x
List of tables	xi
Document revision history	xii
User Guide.....	1
Introduction.....	1
Version history	1
Federate compliance.....	2
Installation	2
System Requirements	2
Installation Instructions	3
Startup.....	4
The command line.....	4
Command line arguments.....	5
Configuration Files.....	5
Sonar configuration file	6
Environment section	6
SonarConfiguration section	7
TrackerConfiguration Section	9
TargetProfiles section	10
Narrow-band auto-detector/signal follower configuration file	13
Configurations	14
Broad-band auto-detector/signal follower configuration file.....	15
Configurations	16
Operation	18
Graphical User Interface	18
Debugging Output	18
Sonar Model	18
Auto-detector / signal follower.....	18
Federate Behaviour.....	19
Time Management Policies	19
Models.....	19
Simulation Object Model.....	19

Execution Management.....	20
Known Faults	20
Known Limitations	20
Bug Reports	21
Technical Description.....	22
Introduction.....	22
Federate Compliance.....	22
Federate Description	23
Functional Description of Models	25
Tracks 25	
Ambient Noise.....	25
The components of ambient noise	25
How the noise beam maps are calculated	26
Transmission loss	27
Simple 27	
Thorpe 27	
Java Classes.....	28
AmbientNoise	28
BBSignal 28	
Complex 28	
ComplexMatrix.....	28
CoordinateConverter.....	28
FFT 28	
PassiveSonarModel.....	28
Signal 29	
Sonar 29	
SonarSystem	29
SonarTarget.....	29
SonarTrack.....	29
TargetData	29
TransmissionLossTable	29
Integration and Testing	30
Integration	30
Long-run.....	30
Future Development.....	31
Related to sonar modeling.....	31
Related to software development	31
Utility class	31
Complex class	31
ComplexMatrix class	31
PassiveSonarModel class.....	32

Sonar	32
SonarSystem	32
TargetData	32
Transmission Loss Tables.....	32
References	33
Annex A. Sample sonar configuration file.....	34
Annex B. Sample narrow-band tracker DLL configuration file	37
Annex C. Sample broad-band tracker DLL configuration file	39
Annex D. Transmission loss table file format and example file	41
Annex E. Noise data file format, conversion and usage	43
E.1 Conversion	44
E.2 Usage by Sonar 3.....	44
Annex F. UML class diagram	45
List of symbols/abbreviations/acronyms/initialisms	46
Distribution list.....	47

List of figures

Figure 1, a sample narrow-band signature.	11
Figure 2, a sample broad-band signature.....	12

List of tables

Table 1, document revision history.	xii
Table 2, federate compliance.....	2
Table 3, system requirements.	2
Table 4, summary of the federate's simulation object model (SOM).	19

Document revision history

Table 1, document revision history.

DATE	VERSION	SUMMARY OF CHANGES
1 April 2007	1.0	First release, covering version 3.0.1 of the Sonar federate.

User Guide

Introduction

Sonar 3.0 is the successor to DRDC – Atlantic’s previous sonar federates and represents a significant increase in fidelity. It is compliant with VMSA 3.0.0 and models passive sonar systems coupled with an auto-detector DLL.

The federate was developed in Java (SDK 1.4.2) using JBuilder Personal Edition 8.0, and Eclipse 3.0 M5. The array simulation was developed by Dr. Bill Roger in IDL [4][5] and ported to Java by Dave Hackett and Allan Gillis. The FFT class was ported from the Pascal code developed by Nils Haeck [9] and available under the Mozilla Public License 1.1. The complex number classes were developed by Dave Hackett and Brad Dillman [7].

The auto-detector/signal follower DLL was developed by Fred Campaigne in ANSI C++ using the Borland compiler. The algorithms used by the detector were supplied by Dr. Bill Roger [5].

The federate code was based on that of Sonar 2 which traces its lineage to the TMA federate written by Jason Murphy and the original Sonar federate written by Greg Denehy [8].

Version history

Version 1.0 was developed between 5 February 2003 and 2 May 2003. It included only a basic simulation of broad-band using the Sonar equations [1].

Version 2.0 was completed on 11 July 2003 and included narrow-band simulation, ambient noise calculations, and transmission loss tables. This is also a sonar equation model [1].

Version 3.0 development commenced in the fall of 2003 and finished in July 2004. This version includes much more realistic modelling of the sonar system as a passive towed-array (although it can be used for other types of systems). **Note:** The ability to model more than one Sonar system with a single federate is lost in this version; every sonar system must now be modelled with its own instance of Sonar 3.

Version 3.0.1 fixed some bugs found in testing, in particular with the transmission loss tables.

Federate compliance

Table 2, federate compliance.

ITEM	VERSION	COMMENTS
Sonar Federate	3.0.1	Passive broad and narrow band tracks, beam forming.
Programming language	Java SDK 1.5	Developed with the Eclipse IDE.
RTI	DMSO 1.3NGv4, DMSO 1.3NGv6, and MäK RTI 2.4.	DMSO Java binding DLL must be present in the PATH for inter-operability with the MäK RTI.
Platform	Windows 2000, XP	Although the federate code is written in Java, the auto-detector/signal follower is a Windows only DLL.

Installation

System Requirements

Sonar 3 is a Windows only federate due to the DLL used for auto-detection and signal following. Since it generates beam maps the memory and processing requirements increase as the number of beams increases and the frequency separation (for narrow-band) decreases.

The federate depends on several third-party JAR files which are included in the federate release. The auto-detector/signal follower DLL relies on several DLLs which are part of Borland's set of libraries; these are also included.

No attempt to determine the minimum system requirements has been made, but the lowest-end machine it has been run on to date is show in Table 3, system requirements.

Table 3, system requirements.

ITEM	MINIMUM REQUIRMENT
Make and model	Dell Latitude D800
CPU	Intel Centrino 1.7GHz
Memory	1.0 GB DDR RAM
Network connection	10/100 Switched Ethernet
Operating System	Windows 2000 Professional

This configuration was capable of handling beam maps of 43 beams x 2000 frequency bins with a single target in real time. Larger beam maps, multiple sonar systems, or scenarios with many targets may require a more powerful configuration.

Installation Instructions

The installation zip file includes all the required executables, DLLs, JAR files, and documentation in their correct sub-directories. Simply unzip the file to the “federates” directory on your system. For example, in the Virtual Combat Systems at DRDC-Atlantic you would unzip the installation file to:

V:\DRDC-Simulations\apps\vm\sa\federates

This will create the following directory structure:

V:\DRDC-Simulations\apps\vm\sa\federates\sonar-3.0.1-vm\sfom-3.0.0

|-> bin

|-> doc

|-> lib

|-> src

“Bin” holds the configuration files, start batch file, signal follower and related DLLs, and any out-put files for results and debugging.

“Doc” holds the Javadoc HTML files that describe the source code API, as well as the user guide and this document (as PDF).

“Lib” holds all required JAR files, including the one for Sonar.

“Src” holds the Java source code for Sonar and the JNI portions of the code for the signal follower DLL.

At the time of writing, a batch-file system is used at DRDC-Atlantic to run federations. Since there are many ways to start a federation the entire process is not described here. However, there are several command line arguments and environment variables that need to be set, and these are described below in the “Start-up” section.

Startup

Starting a Sonar 3 federate is normally done with a set of batch files. There are several systems for doing this, but this document will only refer the method currently in use at DRDC Atlantic. Sonar 3 is started in the same way as other non-GUI federates and it should be easy to adapt these instructions to another system.

The command line

The command line needed to run Sonar 3 is:

```
java -Xms256m -Xmx512m -DVMSBC_ROOT=%VMSBC_ROOT%  
-DVMSBC_JAVA_VERSION=%VMSBC_JAVA_VERSION%  
-classpath %SONAR_CLASSPATH% Sonar.Sonar %CMD_LINE_ARGS%
```

Where, all the options should appear on a single line in your batch file. The environment variables %VMSBC_ROOT%, %VMSBC_JAVA_VERSION%, %SONAR_CLASSPATH%, and %CMD_LINE_ARGS% are set using the system of batch files that runs the command line. The arguments are explained below:

- **Xms256m** : this argument is optional and tells the java virtual machine that it should allocate 256MB of memory to it's pool on start-up. This flag was required only when using beam maps with 2000 frequency bins. The amount of memory to allocate can be changed by altering the number; for example -Xms128m would allocate 128MB of memory. For more information see Sun's 1.4.2 SDK documentation.
- **Xmx512m** : this is also an optional argument, and tells the Java VM that it should not use more than 512MB of memory in it's allocation pool. This option is not usually required, and could lead to "Out of Memory" errors if large beam maps are used.
- **DVMSBC_ROOT** : this argument specifies the root directory for the VMSBC base-classes, which are used by all Java VMSA federates.
- **DVMSBC_JAVA_VERSION** : this argument specifies which version of the VMSBC base classes are required. For this version of Sonar, version 1.0.2 is required.
- **Classpath** : the Java class path needed to include all the libraries for Sonar.
- **Sonar.Sonar** : the package and class for the entry point to the Sonar application. In the case of Sonar 3 the package is called "Sonar", as is the main class. The %CMD_LINE_ARGS% environment variable is the set of command line arguments that are passed to Sonar.

Command line arguments

Sonar 3 takes a number of command-line arguments. These are:

- **-fx** : the name of the federation. It must be set or Sonar will not start.
- **-fn** : the name of the federate. This must be set or Sonar will not start.
- **-fed** : the name of the FED file to use. This is optional and the default is “vmsfom-3.0.0.fed”.
- **-som** : the name of the SOM file to use. This is optional, and the default is “SonarSOM.xml”.
- **-d** : the debug level. This is optional and the argument is any arbitrary number. Sonar does not make much use of this flag, but the VMSA base-classes do report debug information based on its value. In general, the higher the number, the more debug information you will see.

An example command line argument string is:

```
-fx SONAR3_TEST -fn Sonar -fed c:\federation\vmfom-3.0.0.fed -d 0
```

Configuration Files

There are three configuration files used by this federate. The first specifies information and options for the Java Sonar simulation, the other two give parameters for narrow-band and broad-band operation of the auto-detector/signal follower DLL. The DLL configuration files are passed to it by Sonar 3 and are parameters of Sonar’s configuration file. The VMSA execution manager, VMSEM, passes sonar’s configuration file name to it.

It is good practice to include the full path as part of the name. If the full path is not included the federate will attempt to load the configuration files from the current working directory. This can lead to confusion, especially if more than one federation is being configured.

All configuration files are XML and schemas are provided for validation. These schemas should be used with a good XML editor to reduce errors in the configuration. While the schemas cannot catch every possible error they do prevent missing elements and out-of-range values. They are also included in the “XML_Schemas” directory of the installation.

Sonar configuration file

The schema for validating Sonar configurations is “Sonar3Configuration.xsd”, and you will find it in the “XML_Schemas” directory of your Sonar 3 installation. An example configuration file is included as Appendix A.

Sonar configuration files have four sections:

- **Environment:** contains all environmental parameters; sound speed, ocean depth, etc.
- **SonarConfiguration:** has all the information needed to describe the sonar system.
- **TrackerConfiguration:** contains the configuration file names for the DLL, and other parameters for the Java simulation that are directly related to the operation of the DLL.
- **TargetProfiles:** this section has acoustic signature information for the various composite entities in the simulation.

Environment section

The environment section **MUST** have the following information:

SoundSpeed: the speed of sound in the water, in metres/second. The sonar model does not use profiles, only a single average value.

OceanDepth: the average depth of the ocean at the scenario location, in metres. This value is only used for calculating transmission loss by the Thorpe method.

TransmissionLossMethod: the method to use for calculating transmission loss. There are 3 possibilities: Simple, Thorpe, and Table Lookup. The methods are described fully in the Technical Description section.

TransmissionLossTable: if the transmission loss method is “TableLookup” then you need at least one of these. The element has 3 attributes:

- **MinFreq:** the minimum valid frequency for the data in the table.
- **MaxFreq:** the maximum valid frequency for the data in the table.
- **FileName:** the name of the file containing the table (including the path).

The transmission loss file format is given in Appendix D.

AmbientNoiseMethod: the method to use for determining ambient noise levels. There are two options: Static, and Calculate. If Static is chosen then a single level is used for every frequency.

If calculate is used then the noise is calculated as described in the Technical Description section. All of the tags used with the “calculate” method are ignored if the “static” method is selected.

Depending on the choice of ambient noise method there are four optional pieces of information that might be needed in the environment section. If the “Static” ambient noise method is chosen then you **must** also give:

AmbientNoiseLevel: the level, in dB, for ambient noise at all frequencies. This is ignored if the method is “Calculate”.

If the “Calculate” method is chosen the following information **must** be in the file:

ShippingLevel: a parameter used to characterize distant shipping traffic. The valid values are integers from 1 to 9 and the sound level increases with shipping level.

SeaState: this is used when calculating the contribution from surface noise. The valid range is integers from 1 to 7. Higher sea states result in more noise.

RainState: a parameter that describes the amount of rain falling. The valid values are (in order of increasing noise) NoRain, Intermediate, Moderate, and Heavy.

There is one optional tag that may be included (once) when using the “calculate” method:

NoiseFile: this is used to define a noise data file that will be used to generate noise beam maps. The tag has 4 attributes:

- **Name:** the name of the file, including the path.
- **nBeams:** the number of beams per record.
- **nFrequencies:** the number of frequency bins per record.
- **nRecords:** the number of records in the file.

The file format for noise files is described in Appendix E.

SonarConfiguration section

All information for this section **must** be present:

SonarSystem: this tag has two attributes, “Name”, and “Model”. The name is the VMSA object name for sonar system component entities, and the model value must be set to “Passive”. This information is not really required anymore and this tag will probably be removed in the next incremental version of Sonar.

Broad Band: this is the information required to set the frequency range for broad band operation. The XML tag has two attributes: Frequency and Bandwidth. The frequency attribute is the central frequency for the band, in Hz. The Bandwidth attribute is the total range of interest, also in Hz. The total range used is Frequency – ½ Bandwidth to Frequency + ½ Bandwidth. ***Note: it is very important that the lowest frequency is not zero or less.***

MinFrequency: the lowest frequency that will be considered for narrow-band operation, in Hz.

MaxFrequency: the highest frequency that will be considered for narrow-band, in Hz.

IgnoreOwnship: a flag to say if the sonar's parent ship should be included in the signals added to the beam map; true or false. ***Note: there is a bug in version 1.2 of the VMSA Java base classes that breaks the signal processing in Sonar 3 when this tag is set to "false". Until this bug is corrected IgnoreOwnship must be set to "true" only!***

WatchSide: the side to look at. Valid options are "Both", "Port", and "Starboard". For a towed array you should use both, but when simulating a flank array you should pick the corresponding side. Remember: to simulate a flank array, run two separate sonar federates, each with a different WatchSide. Targets not on the given side of the SonarSystem will not be processed and added to the beam map. ***Note: this parameter MUST match the "Side" value in the tracker configuration file or else the PassiveSonarTrack objects will be created with the wrong bearing value.***

TimeStep: for HLA time management Sonar has a 1 second look-ahead, but the beam maps do not have to be updated at the same frequency. The time step causes beam map updates to happen on multiples of the look-ahead. For example, if the time step is 4 the beam maps will only be calculated every 4 simulation seconds.

TotalBeams: the total number of beams produced by the array.

Averages: the signal follower DLL can be set to average successive beam maps together. This improves signal detection. This parameter tells the DLL how many beam maps to average together; the minimum is 1 (no averaging).

NumSensors: the number of sensors in the array.

SensorSeparation: the distance between sensors, in metres.

ForwardBeamOffset: some towed arrays allow you to steer the forward beam. This parameter allows you to say how far off center it should point, in degrees.

CenterBeamAngle: the angle the central beam is pointing, in degrees.

SignalBearingSigma: used to introduce some variability in the bearing when applying signals to the beam maps. The value only affects narrow-band processing.

StartWindowBeam: if you want the DLL to only consider part of the beam map this sets the start of the beam range. It is an integer between 0 and Total Beams – 1. Normally you should set this to 0.

NumWindowBeams: the number of beams to use if you only want the DLL to look at part of the beam map. Normally this should be set to the same number as Total Beams.

StartWindowFreq: the starting frequency bin for the DLL when only considering a section of the beam map. For normal use this should be set to 0.

NumWindowFreqs: how many frequency bins the DLL should consider when considering only a section of the beam map. Normally this should be set to the total bandwidth / frequency separation.

BeammapFileName: for debugging or analysis purposes you sometimes need to have a record of the beam maps that have been processed by the DLL. If you choose to print-out any of these maps they are stored in this file. If the file exists it is over-written, not appended to. These files can be very large (especially when operating in narrow-band).

PrintNoiseBeammap: print-out the noise only beam maps; true or false.

PrintSignalBeammap: print-out the signal only beam maps; true or false.

PrintTotalBeammap: print out the summed noise and signal beam maps; true or false.

PrintNormalisedBeammap: print-out the normalized beam maps; true or false.

TrackerConfiguration Section

All information in this section **MUST** be present.

PrintTracks: a flag to say if track information should be printed to STDOUT (the console window).

GenerateTrackType: what kind of tracks should we look for? “BroadBand”, “NarrowBand”, or “Both”.

TrackerBBConfigFile: the name of the broad-band configuration file for the signal follower. A file with this name must exist in the federate’s working directory.

TrackerNBConfigFile: as above, but for narrow-band.

TargetProfiles section

This section must be present in the configuration file and **must** have at least one target defined. Each target must have an entry in this section if it could potentially be heard by sonar. You can define as many target profiles as you like (each profile is the acoustic signature). At run time these are assigned by matching the “Name” attribute of the “Target” element to CompositeEntity “Name” attributes. If a composite entity cannot be matched to any target profiles then it cannot be detected by Sonar.

Target: each of these tags gives a passive acoustic signature to set of composite entities. The tag has a single attribute and contains two child tags. The attribute is:

- **Name:** the composite entity names that should use this profile. The name may be very specific or very general. For example, a value of “SeaSurface.Military.Warship” will match any composite entity name starting with that string, while “SeaSurface.Military.Warship.Frigate.Australia.FFG.Adelaide Class.HMAS Adelaide” will only match that one specific composite entity.

Each “Target” tag **must** have at least one “Signal” and at least two “BBSignal” child tags.

Signal: these tags are used to characterize the signature for **narrow-band** processing. At least one, but usually three or more, of these “tones” define the signature. Signal tags have three attributes:

- **Frequency:** the central frequency in Hz.
- **Bandwidth:** the frequency range for which the level is valid, in Hz.
- **Level:** the spectral level of the signal, in dB ref. signal bandwidth.

For example, a frequency of 120 Hz, a bandwidth of 200 Hz, and a level of 112 dB is handled as if the target signal was 112 dB at every frequency from 20 Hz to 220 Hz. The graph in figure 1 shows an exaggerated sample narrow-band signature to illustrate the concept.

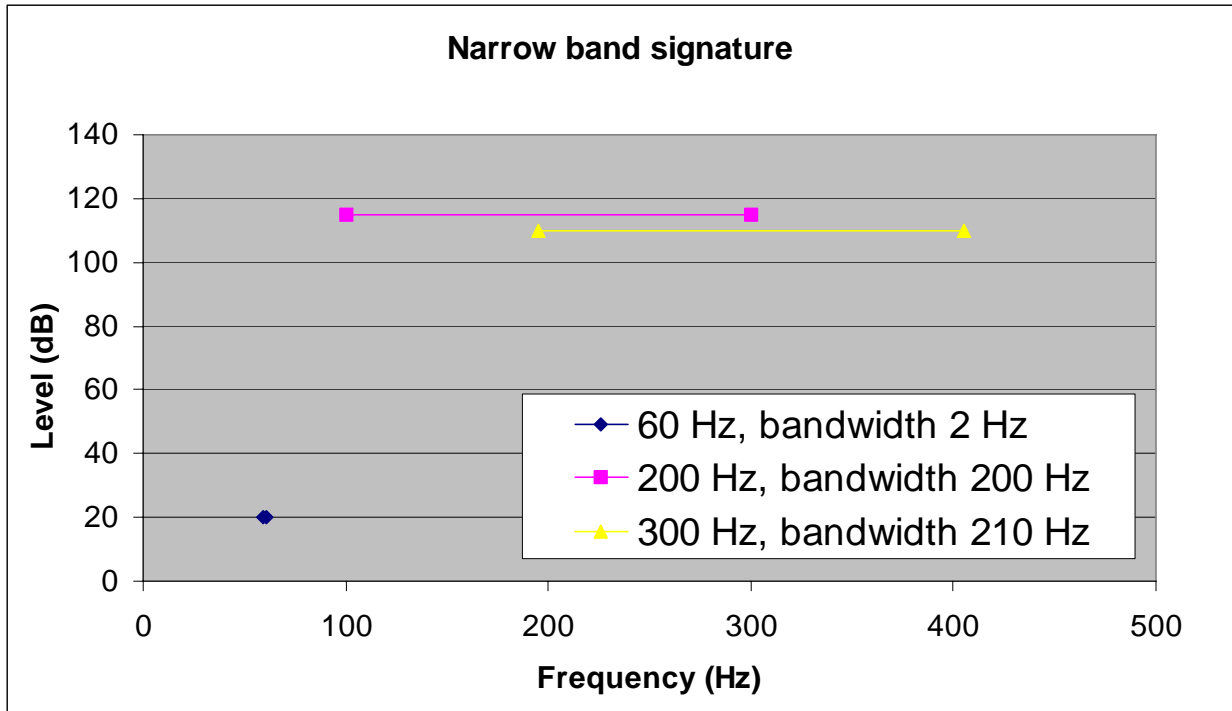


Figure 1, a sample narrow-band signature.

BBSignal: these tags are used for defining the signature for **broad-band** processing. At least two of these must be included as the level for any given frequency is found by interpolation between points. Essentially each BBSignal tag represents a single point on the level v.s. frequency curve defining the signature. These tags have two attributes:

- **Frequency:** the frequency of the point, in Hz.
- **Level:** the spectral level for the point, in dB, ref. a 1 Hz band.

An example signature is shown in figure 2.

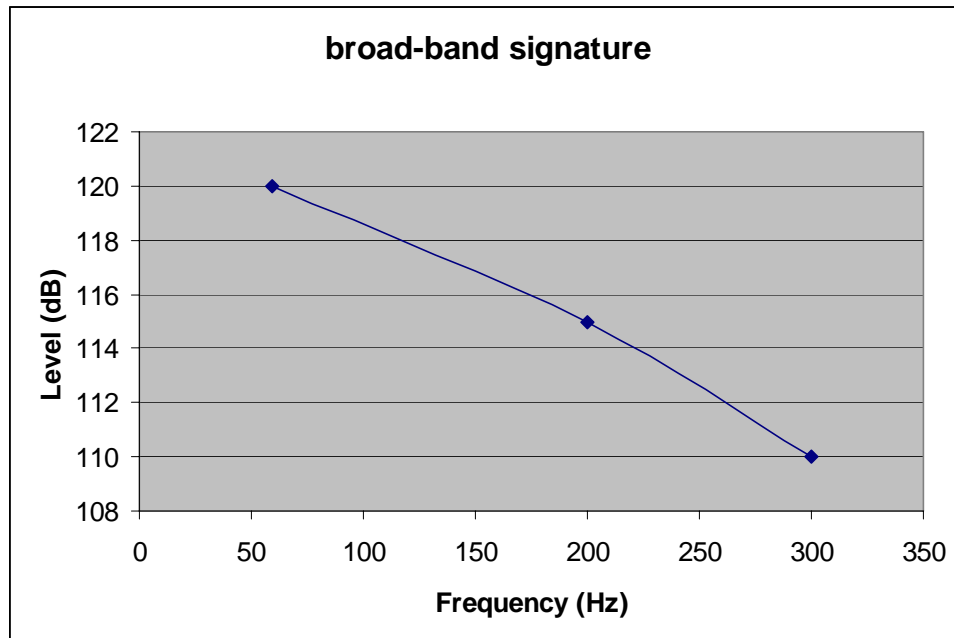


Figure 2, a sample broad-band signature.

In the next version of Sonar these two signature definitions will be consolidated.

Narrow-band auto-detector/signal follower configuration file

The narrow band configuration file schema file is called “SAPSConfigure.xsd”. The parameters for configuration files are detailed below. Note: the beam maps are all calculated in power space, not dB, but this is transparent to the end user.

DataSource: a name to use as a prefix for the log file name.

NoiseDistributionMinimumSNR: the lowest signal to noise ratio (SNR) value in the beam map that should be considered noise. A good typical value to use is 0.1.

NoiseDistributionMaximumSNR: the highest SNR value in the beam map that should be considered noise. A good typical value to use is 15.

HistogramBinWidth: unless you are familiar with the internal workings of the signal follower, you should not adjust this parameter. A good default value to use is 0.1. As part of its internal calculations the signal follower creates a noise distribution which comes from integrating a histogram of the noise. This parameter gives some control over this histogram by indirectly allowing the number of cells to be set. The equation for the number of cells is:

$$(\text{NoiseDistributionMaximumSNR} - \text{NoiseDistributionMinimumSNR}) / \text{HistogramBinWidth}$$

TurnThreshold: this threshold sets the minimum difference in ownship heading that should be considered a turn, in degrees. A good default value to use is 0.1. While in a turn the signal follower does not create new sonar tracks, the noise distribution is not calculated, auto-detection is off, and the variance for signal followers does not grow. When this type of signal follower is used in other applications this parameter is really the difference in heading between the towing ship and the array. With this federate there is never any difference since the ship and array motions are not modeled independently.

BeamMapAverages: part of the beam map processing involves averaging the maps over time to reduce the influence of noise. Set this to any positive integer and that many maps will be averaged.

Trace: “0” for “off”, “1” for “on” (**do not use true/false**). When Trace is on, the DLL will write detailed debugging information to a file in the same directory as the DLL. This file will contain information about the state of every running signal follower for every update cycle. This is really included only for developers working on the DLL and isn’t very useful for end users.

LogResults: “0” for off, “1” for on (**do not use true/false**). When LogResults is on, the DLL writes information about the state of the signal followers for every update cycle to a file in the same directory as the DLL. This file can be useful for analyzing the behaviour of the signal followers, and is at a level that is useful for end users.

NConfigurationElements: the signal followers can run with several configurations. Each configuration must have a configuration section in the file. If you use more than one configuration it is entirely possible that multiple tracks will be generated for any target.

Configurations

Every configuration is defined in it's own section, enclosed in a pair "Configuration" tags. The information inside these sections is:

Delay: this should be set to zero. It allows the DLL to skip a number of update cycles for the auto-detector/signal follower before actually processing the beam maps. This can be useful when using real noise file with known anomalies near its beginning.

PFA: Probability of false alarm. Good values for this parameter are between 0.01 and 0.001, and the allowable range is 0 to 1.0. This parameter influences when signal followers are assigned to potential tracks. The higher the PFA, the more likely a weak signal will be tracked, but the more false tracks will be created.

M: this parameter goes with N. The signal follower only creates a track when it detects a contact M times out of N consecutive updates. For example, an M of 10 with N 12 means the contact must appear 10 times out of 12 consecutive updates before a track is created.

N: see M. Must always be $\geq M$.

BeamWindow: a good initial value for this parameter is one. Over time a signal will move across beams, so the signal follower needs to handle the case where a signal might move into an adjacent beam. This parameter allows the size of the window to be set; the value is in "beams" on either side of centre. For example, if the signal follower was considering a signal in beam 23, a beam window of 1 means that on the next update cycle peaks in beams 22, 23, and 24 will be considered by the follower.

FrequencyWindow: a good initial choice for this parameter is one. A signal may move in frequency over time, as well as in space. Because of this, the signal follower needs to consider adjacent frequency bins when following a signal, just as it needs to consider adjacent beams. This parameter allows the number of bins to either side of the one from the last update cycle to be set.

DetectorAlpha: the detector uses exponential averaging on peaks it finds. This parameter is part of that calculation and should normally be set to 0.1. This is a detail of the internal workings of the DLL that is not generally useful to the end user.

StateAverages: how many values of state variables the signal followers will use to smooth the bearing and frequency rates used in their calculations. The state variables include bearing, bearing rate, frequency, frequency rate, and SNR.

EstimatorNBeams: should normally be set to 5. Since a signal may move across beams between update cycles the signal followers need to be able to look in adjacent beams to find the signal. This parameter lets you set how many beams to either side the followers may consider.

EstimatorNFrequency: should normally be set to 5. Since a signal may also alter in frequency over time the signal followers must be allowed to look at adjacent frequency

EstimatorBounds: in degrees, and good default value to use is 10. When the auto-detector finds a signal it may not be on the best peak in the area of interest. This parameter controls the range of the beam map that the detector will search for a stronger peak before starting up a new signal follower.

LMDThreshold: not used for narrow band. While this parameter must be in the file for it to parse correctly, it isn't actually used by this version of the DLL. It will be removed in future versions.

EndFireOffset: should be set to 5. This parameter is an adjustment that is used to move the bearing of signals directly ahead or astern of the sonar system. This is necessary as the processing algorithms make use of $1/\sin(\text{arrival angle})$ in several places, so there is a singularity for these signals. The value is in degrees.

PixelsFromCentre: not used. While this parameter must be in the file for it to parse correctly, it isn't actually used by this version of the DLL. It will be removed in future versions.

HertzPerPixel: not used. While this parameter must be in the file for it to parse correctly, it isn't actually used by this version of the DLL. It will be removed in future versions.

NSignalEstimatorAverages: not used. While this parameter must be in the file for it to parse correctly, it isn't actually used by this version of the DLL. It will be removed in future versions.

Side: this parameter tells the signal follower which side the sonar is watching. It can be set to "Port", "Starboard", and "Both" by choosing the number 0 for port, 1 for starboard, and 2 for both. This value **must** agree with the "WatchSide" tag value in the Sonar configuration file or the signal follower will not perform properly. Internally the signal follower always creates two PassiveSonarTrack objects per detection, this flag simply tells it to drop the known spurious side.

***Note:** the signal follower knows nothing about how the beam map was created, so this value MUST match the "WatchSide" tag in the Sonar configuration file or else the PassiveSonarTrack objects will have an incorrect bearing. For example, if the Sonar configuration specified "Both", targets from port and starboard will be processed and added to the beam map. If the signal follower is set to "Port" it will still detect targets from both sides because they were added to the beam map, but all tracks will be created with the port bearing and the starboard bearing will be discarded.*

Broad-band auto-detector/signal follower configuration file

The broad-band configuration file schema file is called "SAPSConfigure.xsd". The parameters for configuration files are:

DataSource: a name to use as a prefix for the DLL's log and debug file names.

TurnThreshold: the signal follower does not create new sonar tracks while the parent ship is in a turn. This threshold sets the minimum heading rate that should be considered a turn, in degrees/second. With the current motion model any heading variation really is a turn, but in the future more detailed motion models may cause heading variations unrelated to turning the ship. In testing a setting of 0.1 worked well.

BeamMapAverages: part of the beam map processing involves averaging the maps over time to reduce the influence of noise. Set this to any positive integer and that many maps will be averaged.

Trace: “0” for “off”, “1” for on (**do not use true/false**). When Trace is on the DLL will write detailed debugging information to a file in the same directory as the DLL. This file will contain information about the state of every running signal follower for every update cycle. This is really included only for developers working on the DLL and isn’t very useful for end users.

LogResults: “0” for “off”, “1” for “on” (**do not use true/false**). When LogResults is on the DLL writes information about the state of the signal followers for every update cycle to a file in the same directory as the DLL. This file can be useful for analyzing the behaviour of the signal followers, and is at a level that is

NConfigurationElements: the signal followers can run with several configurations. Each configuration must have a configuration section in the file.

Configurations

Every configuration is defined in it’s own section, enclosed in a pair “Configuration” tags. The information inside these sections is:

Delay: this should be set to zero. It allows the DLL to skip a number of update cycles auto-detector/signal follower before actually processing the beam maps. This can be useful when using real noise file with known anomalies near it’s beginning.

DetectorThreshold: must always be set to 1. This parameter is meant for the trackers when dealing with the general case of broad-band signals. Sonar 3 handles it in a specific way, and this parameter must be set to 1 for the federate to function correctly.

M: this parameter goes with N. The signal follower only creates a track when it detects a contact M times out of N updates. For example, an M of 10 with N 12 means the contact must appear 10 times out of 12 updates before a track is created.

N: see M. This must always be $\geq M$.

BeamWindow: a good initial value for this parameter is one. Over time a signal will move across beams, so the signal follower needs to handle the case where a signal might move into an adjacent beam. This parameter allows the size of the window to be set; the value is in “beams” on either side of centre. For example, if the signal follower was considering a signal in beam 23, a beam window of 1 means that on the next update cycle peaks in beams 22, 23, and 24 will be considered by the follower.

DetectorAlpha: the detector uses exponential averaging on peaks it finds. This parameter is part of that calculation and should normally be set to 0.1. This is a detail of the internal workings of the DLL that is not generally useful to the end user.

StateAverages: how many values of state variables the signal followers will use to smooth the bearing and frequency rates used in their calculations. The state variables include bearing, bearing rate, frequency, frequency rate, and SNR.

EstimatorNBeams: should normally be set to 5. Since a signal may move across beams between update cycles the signal followers need to be able to look in adjacent beams to find the signal. This parameter lets you set how many beams to either side the followers may consider.

EstimatorBounds: in degrees, and good default value to use is 10. When the auto-detector finds a signal it may not be on the best peak in the area of interest. This parameters controls the range of the beam map that the detector will search for a stronger peak before starting up a new signal follower.

LMDThreshold: must always be set to 1. This parameter is meant for the detector when dealing with the general case of broad-band signals. Sonar 3 handles it in a specific way, and this parameter must be set to 1 for the federate to function correctly.

EndFireOffset: should be set to 5. This parameter is an adjustment that is used to move the bearing of signals directly ahead or astern of the sonar system. This is necessary as the processing algorithms make use of $1/\sin(\text{arrival angle})$ in several places, so there is a singularity for these signals. The value is in degrees.

NSignalEstimatorAverages: not used. While this parameter must be in the file for it to parse correctly, it isn't actually used by this version of the DLL. It will be removed in future versions.

Side: this parameter tells the signal follower which side the sonar is watching. It can be set to "Port", "Starboard", and "Both" by choosing the number 0 for port, 1 for starboard, and 2 for both. This value **must** agree with the "WatchSide" tag value in the Sonar configuration file or the signal follower will not perform properly. Internally the signal follower always creates two PassiveSonarTrack objects per detection, this flag simply tells it to drop the known spurious side.

***Note:** the signal follower knows nothing about how the beam map was created, so this value MUST match the "WatchSide" tag in the Sonar configuration file or else the PassiveSonarTrack objects will have an incorrect bearing. For example, if the Sonar configuration specified "Both", targets from port and starboard will be processed and added to the beam map. If the signal follower is set to "Port" it will still detect targets from both sides because they were added to the beam map, but all tracks will be created with the port bearing and the starboard bearing will be discarded.*

Operation

Graphical User Interface

There is no GUI for this federate.

Debugging Output

There are three different sources for debugging info; the federate and base classes, the auto-detector / signal follower DLL, and the Sonar model itself. The federate and base class debugging is controlled with the command line flag “-d x” and is fairly common amongst VMSA federates; the higher the value of ‘x’ the more debug information is produced. Please see the base class Javadocs located in the doc directory of the federate distribution for more information.

Sonar Model

The sonar model allows you to print various beam maps to a text file. This is very useful for determining problems with detection or for comparing to detected tracks. The configuration file controls which (if any) beam maps will be printed (see section 3.3.1 for details). The maps that can be printed are:

- Noise only,
- Signals only,
- Noise and signals summed,
- The final normalized beam map.

The file for these maps grows quickly as data is written to it on every iteration of the simulation loop. This is especially true when operating in narrow-band where beam maps can contain thousands of frequency bins per beam.

Auto-detector / signal follower

While the auto-detector / signal follower DLL does provide debugging output, it is meant for development use only. The output is neither meant for, nor particularly useful for, the end use of this federate and is not documented here.

Federate Behaviour

Time Management Policies

Sonar is both time regulated and time constrained. The look-ahead is hard-coded to 1 second to ensure the positions are updated regularly, but you can change the time between calculating beam maps by using the <TimeStep> tag in the configuration file.

Models

Simulation Object Model

Table 4, summary of the federate's simulation object model (SOM).

CLASSES	PUBLISH	SUBSCRIBE
CompositeEntity		
SeaSurface		X
SubSurface		X
ComponentEntity		
SonarSystem	X	
Track		
RelativeSonarTrack	X	
INTERACTIONS		
ExecutionManagment	X	X
InitialiseFederate	X	X
CreateEntity	X	X
ExecutionManagementError	X	X
TerminateIteration	X	X
TerminateFederation	X	X
RemoveEntity	X	X

Execution Management

Sonar systems need to be defined as ComponentEntities in the execution manager script files. A sample script file for the VMSEM execution manager is below:

```
<ComponentEntity Type="SensorSystem.SonarSystem"
Name="Sub_01_Sonar_01" FederateName="Sonar"
ConfigFile="narrow_config.xml">
  <ComponentEntityAttribute AttributeName="ComponentName">
    <AttributeEntry AttributeValue="SensorSystem.SonarSystem..."
AttributeType="s"/>
  </ComponentEntityAttribute>
</ComponentEntity>
```

The name attribute of the opening <ComponentEntity> tag will be used when naming tracks, so it **must** be unique. The convention at DRDC Atlantic has been to name it with the parent name as a prefix (as shown above: Sub_01_Sonar_01).

Since every sonar system **MUST** be simulated with a separate federate the “FederateName” attribute of the <ComponentEntity> tag must be unique. For example if we were simulating a flank array sonar on a submarine that submarine will have two ComponentEntity entries; the first one could use a federate name of “Sonar_sub1_portside” and the second “Sonar_sub1_starboardside”.

The “ConfigFile” attribute is required and is what the federate uses to find its configuration file. A file of this name must exist in the working directory of the federate.

Known Faults

Setting a window of interest in the signal follower DLL has not been used or tested to date.

The random number seed does not get set correctly from the VMSEM script file. However, we have been unable to get even the VMSEM to correctly display a random number seed entered in the script. We have reported this as a bug in the base classes.

There may be a memory leak that is only visible in very long (11 hours+) federation runs. We are still investigating this, but it seems to have no practical impact on simulation.

Known Limitations

Due to the Auto-detector/ signal follower DLL used by Sonar 3 it can only simulate one system at a time. This is a major change from version 2 which can model any number of sonar systems.

Sonar does not do anything with the save and restore synchronization points. The functions are implemented (as they must be when using the VMSA base classes) but are empty.

Bug Reports

Please report any bugs found with this federate to:

Allan Gillis
DRDC Atlantic
allan.gillis@drdc-rddc.gc.ca

Technical Description

Introduction

Sonar 3.0 is the successor to DRDC Atlantic's previous sonar federates and represents a significant increase in fidelity. It is compliant with VMSA 3.0.0 and models passive sonar systems coupled with an auto-detector DLL.

The federate was developed in Java (SDK 1.4.2) using JBuilder Personal Edition 8.0, and Eclipse 3.0 M5. The array simulation was developed by Dr. Bill Roger in IDL [4][5] and ported to Java by Dave Hackett and Allan Gillis. The FFT class was ported from the Pascal code developed by Nils Haeck [9] and available under the Mozilla Public License 1.1. The complex number classes were developed by Dave Hackett and Brad Dillman [7].

The auto-detector/signal follower DLL was developed by Fred Campaigne in ANSI C++ using the Borland compiler. The algorithms used by the detector were supplied by Dr. Bill Roger [5].

The federate code was based on that of Sonar 2 which traces its lineage to the TMA federate written by Jason Murphy and the original Sonar federate written by Greg Denehy [8].

Federate Compliance

The following table summarizes the compliance for this federate.

	Version	Comments
Sonar 3 Federate	3.0	
Programming Language	Java 1.4.2 (signal follower DLL ANSI C++)	
VMS-FOM	3.0.0	
RTI	DMSO V1.3NG4	
Platform	Windows	

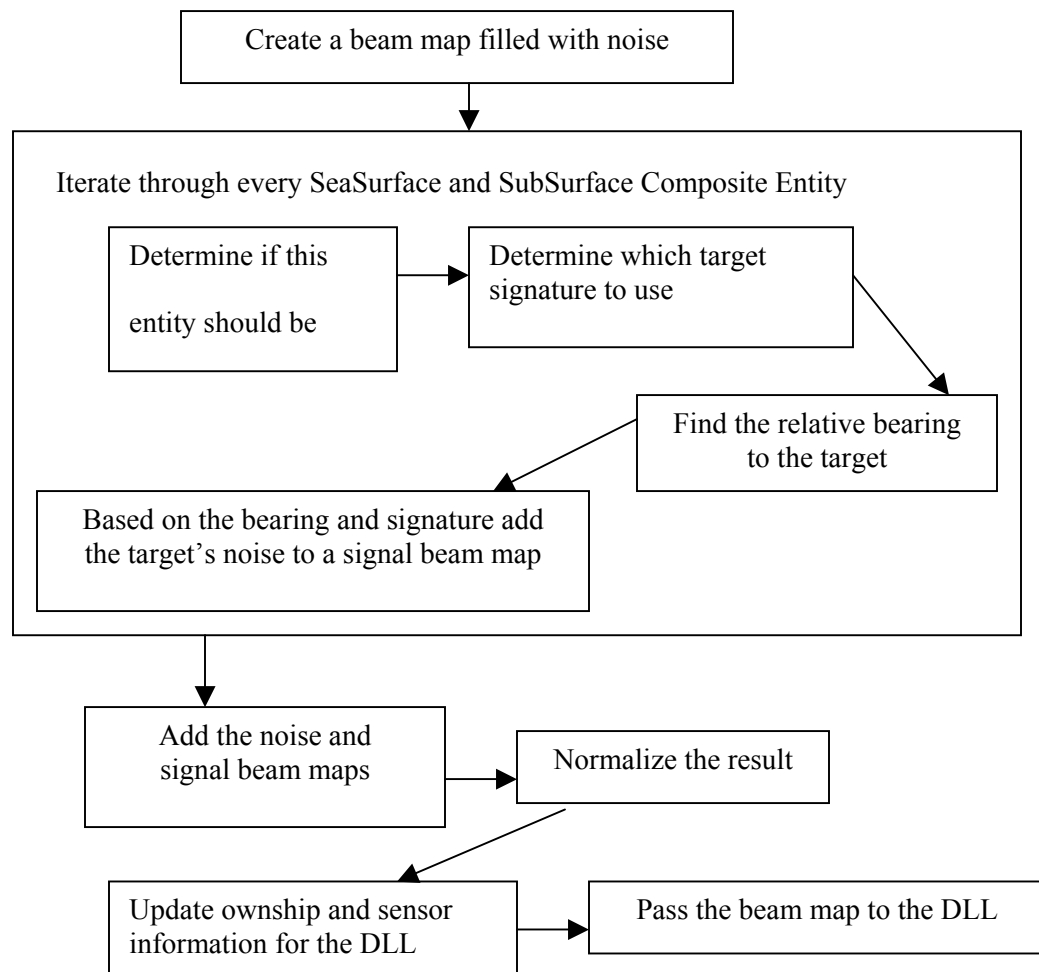
Federate Description

This sonar federate significantly raises the fidelity over that of Sonar 2. It uses signal follower software from Bill Roger (a C++ DLL connected via JNI) to develop tracks from a simulated array beam map. Transmission loss is via Thorpe's [3] equation or a table; noise follows the standard Wentz tables [1].

The signal follower algorithms are the same as those used in The Sonar Test-Bed and are based upon towed array sonar algorithms. The beam is also a representation of a towed array, so in essence this is a towed array sonar model. Turning off the port or starboard signal followers via a configuration file allows it to simulate flank arrays.

While the beam pattern represents a towed array, none of the other characteristics of a towed array are modelled. For example, the towed array doesn't bend, has no length, always follows the same heading as it's parent ship, and is considered to be at the co-ordinate centre of it's parent.

The primary role of the Sonar 3 federate is to create beam-maps for the autodetector/signal follower DLL. The simulation process is almost identical for narrow and broad band, although the resulting beam maps are very different. The process both follow is:



Sonar subscribes to all SeaSurface and Subsurface CompositeEntities and considers every one of them when creating the signal beam map. If you have configured the system to only watch the port side then entities on the starboard side will be skipped, and vice versa. You may also exclude the sonar system's parent entity with the appropriate configuration file option.

Normalization varies between narrow-band and broad-band. For broad band the normalization routine looks for the lowest intensity level in the map and then divides all other cells by that number. This results in broad-band maps with a minimum value of 1.0.

For narrow band normalization is “perfect split-window”, meaning that the average intensity level is computed from the noise map alone. Once the noise and signals are added each cell is divided by this average. This is essentially what a real split-window normalizer does except that a real one doesn't have the luxury of knowing the difference between a signal and noise. The main difference between our normalization and a real split-window normalizer is that the artefacts caused by normalization near a signal will not occur in our system.

Functional Description of Models

Tracks

The auto-detector / signal follower DLL is responsible for creating and updating tracks. As soon as the detector decides it has lost a track it is deleted from the federation simulation and the internal track table.

Ambient Noise

The ambient noise model is based on Wentz's noise equations with corrections from NUSC [2] and DRDC-Atlantic. The noise modeling is handled by two classes: SonarSystem, and AmbientNoise.

The AmbientNoise class is only responsible for generating noise values (in dB) for the various components of ambient noise. The SonarSystem class loads (from the configuration file) all parameters used to calculate the noise components, calls the functions of AmbientNoise to get values for the components, and creates beam maps of the noise. These beam maps are used by PassiveSonarModel to create a total signal and noise beam map.

The components of ambient noise

The ambient noise is computed based on many individual components. These are:

- Turbulence noise,
- Shipping noise,
- Surface noise,
- Thermal noise,
- Rain noise.

For details see the Javadocs which are included in the /doc directory of the federate distribution.

How the noise beam maps are calculated

This version of Sonar operates in both narrow-band and broad-band modes. The broad-band beam map is one dimensional (1 cell per beam) while the narrow-band beam map is two dimensional (1 row per beam, 1 cell per frequency). The number of beams and frequencies are controlled by the configuration file parameters.

The noise beam maps are calculated using a Gaussian distribution. Since the beam maps contain intensity values the dB levels are converted before applying the distribution equation. The equation for noise intensity in any given beam map cell is:

$$N = (\sigma \bullet G)^2$$

Where:

N is the noise intensity,
G is the next number in the normal Gaussian sequence,
 σ is the standard deviation.

The standard deviation comes from the computed ambient noise level by:

$$\sigma = \sqrt{10^{L/10}}$$

Where:

σ is the standard deviation,
L is the computed ambient noise level (dB).

For narrow-band each cell is assigned a single noise value. For broad-band a number of values are added together over the frequency range to get the total noise in each beam. For example, if the broad-band frequency range is 10 Hz to 2500 Hz and 200 frequencies are set (via the configuration file) each beam will have a sum of 200 noise values evenly spaced between 10 and 2500 Hz inclusive.

Transmission loss

Transmission loss is handled in one of three ways:

- **Simple:** a compromise equation for transmission loss as a function of range. There is no frequency dependence or consideration of the transition from spherical and cylindrical spreading.
- **Thorpe:** calculates TL from spherical and/or cylindrical spreading, and absorption [3].
- **Table lookup:** the TL comes from tables of transmission loss vs range. Several tables can be used to represent different slices of frequency. There is no interpolation between frequencies.

In all cases the source and receiver depths are not used.

Simple

The equation for the simple method is:

$$TL = 15 \bullet \text{LOG}_{10}(\text{range})$$

Where:

TL is the transmission loss in dB,

Range is the distance from the source to the sonar system.

Thorpe

This method includes contributions to TL from spreading and absorption. The equation for transmission loss is:

$$TL = \text{spreading loss} + \text{absorption loss}.$$

The spreading loss is calculated by:

$$\text{spreading} = 20 \bullet \text{LOG}_{10}(\text{range})$$

When the range is less than the ocean depth, or:

$$\text{spreading} = 20 \bullet \text{LOG}_{10}(\text{depth}) + 10 \bullet \text{LOG}_{10}(\text{range} / \text{depth})$$

when it is greater.

Where:

spreading is the loss due solely to spreading in dB,

depth is the ocean depth in metres,

range is the distance between the source and the sonar system in metres.

Java Classes

Each class used in the federate is described below. Only a brief description is given, but much more detailed information can be found in the Javadocs which are in the doc directory of the federate distribution.

AmbientNoise

This class holds data, and provides functions for calculating the ambient noise levels in the sea. It does not provide methods to create beam maps; this is handled by the SonarSystem class.

BBSignal

The BBSignal class is used to hold signal information for broad-band calculations. In this federate the only signals we consider are targets, so each instance is effectively a broad-band signature.

Complex

This class represents complex numbers and has many functions for operating on them. This class is used extensively by the beam-forming routines in PassiveSonarModel.

ComplexMatrix

This represents matrices of complex numbers. It also includes the ability to multiply one matrix by another, but no other functions.

CoordinateConverter

This class has utilities for dealing with the coordinate system. In particular it has functions to calculate relative bearings and so on. See the Javadocs for more details.

FFT

This class performs forward and inverse Fast Fourier Transforms for complex number series (defined using the Complex class defined above).

PassiveSonarModel

This class is the back-bone of the federate. It does all the beam-forming, decides which signals to include and which to exclude, and hands-off completed beam maps to the signal follower DLL for processing.

Signal

Signal is used for characterizing narrow-band signatures.

Sonar

This class is the federate part of the simulation. It encapsulates the main() entry point to the application and defines the expected methods as specified by the VMSA architecture. It handles all communications with the RTI.

SonarSystem

This class holds all the information that describes the sonar system and also provides several utility functions. In particular it calculates noise maps and loads all the configuration file settings. The PassiveSonarModel class has a Sonar System and makes heavy use of its data and functions.

SonarTarget

This class is used to represent composite entities, and contains their acoustic signature information.

SonarTrack

This class represents sonar tracks and inherits from PublishedRelativeSonarTrack. Objects of this class are created by the Tracker class when called back by the signal follower DLL.

TargetData

This class is used to hold target signature information for composite entities in the simulation. A name attribute is used to match the target data to composite entities when generating beam maps.

TransmissionLossTable

This class is responsible for loading transmission loss data from a text file. It also provides a function to interpolate the data and give transmission loss for a particular range. The class has data members that specify its frequency range.

Integration and Testing

Integration

Sonar 3 has been tested with the following federates:

- VMSEM 1.8.1
- Motion 2.3.0
- Horizon HLA Plug-in 2.5

In all cases the scenario was very simple; a single target was placed at a range of 5 kilometres. We used the Horizon Helm to steer both ownship and the target.

Sonar 3 does respond correctly to the “End Federation” signal from VMSEM.

Long-run

We also ran an extended test without Horizon participating. This test ran from 16:30 to 07:45 the next morning. It successfully completed 3 loops for a total federation time of about 11 hours. While the Windows 2000 task manager showed available memory decreasing over time there was no perceivable performance problems with the computer. The available memory returned to its pre-simulation level once the federation was ended.

Future Development

While developing and tuning this federate many opportunities for future expansion were identified. These are described below, grouped into two general categories of things that will improve the sonar simulation, and those that will help future software development.

Related to sonar modeling

Ambient noise calculation: it would be better to load the surface noise and shipping noise tables from a file. Right now the only way to change this data is to alter the AmbientNoise class and recompile.

Active sonar: this version only handles passive sonar systems. An extension to active sonar would greatly enhance its usefulness.

Signals: currently narrowband and broadband signals are handled separately. These could be collapsed into a single representation of a target profile. This would simplify the sonar configuration file somewhat, and eliminate a class.

Operator interface: now that Sonar produces beam maps we are in a position to include real GUI elements that could be used by sonar operators. For the next version we are tentatively planning to include LOFAR and FRAZ displays. With these displays an operator could designate contacts manually and also vet the contacts generated by the auto detector.

Related to software development

Utility class

There are several instances where identical functions are defined in many different classes. These functions should be collected into a single utility class, or even split-out into a new reusable package.

Complex class

There are some redundancies in this class' functions. It could use a clean-up and would fit nicely into a new Utility class.

ComplexMatrix class

Another excellent candidate to move into a Utility package or class.

PassiveSonarModel class

There are many functions in the PassiveSonarModel that would be good candidates to put in a utility class. They are:

- DbtoI: converts dB values to intensity.
- ItoDB: converts intensity values to dB.
- MaxInArray: finds the maximum value in a one dimensional array.
- AbsAngles: converts a one-dimensional array to it's absolute values.
- Bessel_I
- COSD
- ACOSD

Sonar

Sonar is currently using both log4j and it's older log file routines. We should consolidate these.

SonarSystem

There are many things in this class that should more properly belong elsewhere. In particular the environmental parameters like noise and sound speed should be broken out into a new class. It also implements its own version of Log10, DbtoI, and ItoDB, and these should be removed and placed in a utility class.

TargetData

Currently the narrowband and broadband signatures are handled separately. This should be consolidated in the next version.

Transmission Loss Tables

The current format is based on output from an old FORTRAN program. This should be migrated to an XML format that includes some meta-data to indicate where the information comes from, and to make the format consistent with the test of the Sonar 3 configuration files.

References

- [1] Urick, Robert J., "Principles of Underwater Sound", 3rd Edition, McGraw Hill, New York, 1983.
- [2] Walt Sadowski, Richard Katz, and Kathleen McFadden, "Ambient Noise Standards for Acoustic Modeling and Analysis," NUSC Technical Document 7265, 3 August 1984.
- [3] Tyler, Gordon D., "The Emergence of Low-Frequency Active Acoustics as a Critical Antisubmarine Warfare Technology", Johns Hopkins APL Technical Digest, Volume 13, Number 1, 1992.
- [4] W.A. Roger, "Evaluation of the Probabilistic Data Association Filter in a Realistic Sonar Environment", DREA TM 94/201, May 1994.
- [5] W.A. Roger, private communication.
- [6] Dave Hackett, unpublished manuscript.
- [7] Brad Dillman and Dave Hackett, unpublished manuscript.
- [8] Greg Denehy, private communication.
- [9] Nils Haeck, "FFT Library", <http://www.simdesign.nl/fft.html>, SimDesign, March 2003.

Annex A. Sample sonar configuration file

```
<SonarConfigurationFile>
  <Environment>
    <AmbientNoiseMethod>Calculate</AmbientNoiseMethod>
    <AmbientNoiseLevel>70</AmbientNoiseLevel>
    <ShippingLevel>5</ShippingLevel>
    <SeaState>5</SeaState>
    <RainState>Moderate</RainState>
    <NoiseFile Name="noise.dat" nBeams="215" nFrequencies="408" nRecords="878"/>
    <TransmissionLossMethod>Simple</TransmissionLossMethod>
    <TransmissionLossTable MinFreq="50" MaxFreq="250" FileName="TL_50_250.sap"/>
    <TransmissionLossTable MinFreq="251" MaxFreq="950" FileName="TL_250_950.sap"/>
    <OceanDepth>2000.0</OceanDepth>
    <SoundSpeed>1500.0</SoundSpeed>
  </Environment>

  <SonarConfiguration>
    <SonarSystem Name="SensorSystem.SonarSystem" Model="Passive"/>
    <TotalBeams>43</TotalBeams>
    <StartWindowBeam>0</StartWindowBeam>
    <NumWindowBeams>43</NumWindowBeams>
    <Averages>1</Averages>
    <StartWindowFreq>0</StartWindowFreq>
    <NumWindowFreqs>201</NumWindowFreqs>
    <NumSensors>48</NumSensors>
    <SensorSeparation>2.0</SensorSeparation>
    <ForwardBeamOffset>0.0</ForwardBeamOffset>
    <CenterBeamAngle>90.0</CenterBeamAngle>
    <WatchSide>Both</WatchSide>
    <IgnoreOwnship>true</IgnoreOwnship>
    <FrequencySeparation>0.1</FrequencySeparation>
    <SignalBearingSigma>2.0</SignalBearingSigma>
    <MaxFrequency>210.0</MaxFrequency>
    <MinFrequency>190.0</MinFrequency>
    <TimeStep>4</TimeStep>
    <BroadBand Bandwidth="2489" Frequency="1245"/>
    <BeammapFileName>beammaps.txt</BeammapFileName>
    <PrintNoiseBeammap>>false</PrintNoiseBeammap>
    <PrintSignalBeammap>>false</PrintSignalBeammap>
    <PrintTotalBeammap>>false</PrintTotalBeammap>
    <PrintNormalisedBeammap>>false</PrintNormalisedBeammap>
  </SonarConfiguration>
</SonarConfigurationFile>
```

```

<TrackerConfiguration>
    <PrintTracks>false</PrintTracks>
    <TrackerNBConfigFile>NBTrial.xml</TrackerNBConfigFile>
    <TrackerBBConfigFile>BBTrial.xml</TrackerBBConfigFile>
    <GenerateTrackType>NarrowBand</GenerateTrackType>
</TrackerConfiguration>

<TargetProfiles>
    <Target Name="SubSurface.Military.Submarine">
        <Signal Frequency="200" Level="115" Bandwidth="200"/>
        <Signal Frequency="300" Level="110" Bandwidth="210"/>
        <Signal Frequency="60" Level="120" Bandwidth="2"/>
        <BBSignal Frequency="10" Level="90"/>
        <BBSignal Frequency="300" Level="90"/>
        <BBSignal Frequency="2500" Level="90" />
    </Target>
    <Target Name="SeaSurface.NonMilitary.FishingBoat">
        <Signal Frequency="150" Level="150" Bandwidth="100"/>
        <Signal Frequency="250" Level="145" Bandwidth="100"/>
        <Signal Frequency="350" Level="140" Bandwidth="100"/>
        <Signal Frequency="60" Level="20" Bandwidth="2"/>
        <BBSignal Frequency="10" Level="90"/>
        <BBSignal Frequency="300" Level="90"/>
        <BBSignal Frequency="2500" Level="90" />
    </Target>
    <Target Name="SeaSurface.Military.Warship">
        <Signal Frequency="400" Level="125" Bandwidth="400"/>
        <Signal Frequency="200" Level="135" Bandwidth="2"/>
        <Signal Frequency="60" Level="120" Bandwidth="2"/>
        <BBSignal Frequency="60" Level="120"/>
        <BBSignal Frequency="200" Level="135"/>
        <BBSignal Frequency="2500" Level="90" />
    </Target>
    <Target Name="SeaSurface.NonMilitary.ContainerShip">
        <Signal Frequency="300" Level="140" Bandwidth="400"/>
        <Signal Frequency="60" Level="120" Bandwidth="2"/>
        <BBSignal Frequency="10" Level="90"/>
        <BBSignal Frequency="300" Level="90"/>
        <BBSignal Frequency="2500" Level="90" />
    </Target>
    <Target Name="SeaSurface.NonMilitary.Liner">
        <Signal Frequency="300" Level="140" Bandwidth="400"/>
        <Signal Frequency="60" Level="120" Bandwidth="2"/>
    </Target>

```

```
        <BBSignal Frequency="10" Level="90"/>
        <BBSignal Frequency="300" Level="90"/>
        <BBSignal Frequency="2500" Level="90" />
    </Target>
</TargetProfiles>
</SonarConfigurationFile>
```

Annex B. Sample narrow-band tracker DLL configuration file

```
<?xml version="1.0"?>

<NarrowBandConfiguration xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance"
xsi:noNamespaceSchemaLocation="http://wallaby/xmldocs/SAPSConfigure.xsd
">

    <DataSource>SH5_PARTA_HEND_5RA.drea_V2b</DataSource>

    <NoiseDistributionMinimumSNR>0.1</NoiseDistributionMinimumSNR>

    <NoiseDistributionMaximumSNR>8.0</NoiseDistributionMaximumSNR>

    <HistogramBinWidth>0.1</HistogramBinWidth>

    <TurnThreshold>0.1</TurnThreshold>

    <BeamMapAverages>1</BeamMapAverages>

    <Trace>1</Trace>

    <LogResults>1</LogResults>

    <NConfigurationElements>1</NConfigurationElements>

    <Configuration>

        <Delay>0</Delay>

        <PFA>0.1</PFA>

        <M>10</M>

        <N>12</N>

        <BeamWindow>2</BeamWindow>

        <FrequencyWindow>2</FrequencyWindow>

        <DetectorAlpha>0.1</DetectorAlpha>

        <StateAverages>10</StateAverages>

        <EstimatorNBeams>7</EstimatorNBeams>

        <EstimatorNFrequency>7</EstimatorNFrequency>
```

```
<EstimatorBounds>10.0</EstimatorBounds>

<LMDThreshold>1.0</LMDThreshold>

<EndFireOffset>5.0</EndFireOffset>

<PixelsFromCentre>2</PixelsFromCentre>

<HertzPerPixel>0.1</HertzPerPixel>

<NSignalEstimatorAverages>1</NSignalEstimatorAverages>

<Side>2</Side>

</Configuration>

</NarrowBandConfiguration>
```

Annex C. Sample broad-band tracker DLL configuration file

```
<?xml version="1.0"?>

<BroadBandConfiguration xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance"
xsi:noNamespaceSchemaLocation="http://wallaby/xmldocs/SAPSConfigure.xsd
">

    <DataSource>VMSA</DataSource>

    <TurnThreshold>0.1</TurnThreshold>

    <BeamMapAverages>1</BeamMapAverages>

    <Trace>1</Trace>

    <LogResults>1</LogResults>

    <NConfigurationElements>1</NConfigurationElements>

    <Configuration>

        <Delay>0</Delay>

        <DetectorThreshold>1.0</DetectorThreshold>

        <M>4</M>

        <N>5</N>

        <BeamWindow>2</BeamWindow>

        <DetectorAlpha>0.1</DetectorAlpha>

        <StateAverages>5</StateAverages>

        <EstimatorNBeams>7</EstimatorNBeams>

        <EstimatorBounds>10.0</EstimatorBounds>

        <LMDThreshold>1.0</LMDThreshold>

        <EndFireOffset>5.0</EndFireOffset>

        <NSignalEstimatorAverages>1</NSignalEstimatorAverages>

        <Side>2</Side>
```

```
</Configuration>  
</BroadBandConfiguration>
```

Annex D. Transmission loss table file format and example file

Transmission loss files are plain text with a very specific format. They were originally generated with a FORTRAN program and the format reflects this.

Each file is valid over a specific spectrum. This spectrum is **not** found in the files and **must** be specified in the “TransmissionLossTable” tags in the Sonar configuration file. Any number of tables can be used. Please see the configuration file section for more information.

The first part of the transmission loss file is a header that has no meaning to Sonar, but describes where the data originally came from. There can be any number of lines in the header.

The last line before the data **must** have the single word “CURVE” on it, starting in the first column. This line separates the header from the loss v.s. range data.

Each line below the “CURVE” line contains two values: range from receiver to target in metres, and transmission loss at that range in dB. The range value **must** start in column 7 and the last digit for range **must** be in column 13. The loss value **must** start in column 20 and the last digit for loss **must** be in column 26.

An example noise file is shown below:

Label 1

Transmission Loss curve
CURVE

2from file C:\My Documents\tri98_150_50.tl

250.000	44.5000
500.000	47.7000
750.000	49.6400
1000.00	51.0700
1250.00	52.2100
1500.00	53.1700
1750.00	54.0000
2000.00	54.7400
2250.00	55.4100
2500.00	56.0100
2750.00	56.5700
3000.00	57.0900
3250.00	57.5700
3500.00	58.0300
3750.00	58.4600
4000.00	58.8600
4250.00	59.2500
4500.00	59.6100
4750.00	59.9600
5000.00	60.3000
5250.00	60.6200
5500.00	60.9200
5750.00	61.2200
6000.00	61.5000
6250.00	61.7800
6500.00	62.0400

Annex E. Noise data file format, conversion and usage

The noise data files are binary, big endian, 32 bit floating point numbers. They are **not** self describing. Each file has a number of records; each record is comprised of beams, and each beam has some number of frequency bins.

The arrangement of the data is:

```
Record 1
    Beam 1
        Frequency bin 1 ... Frequency bin N
    Beam 2
        Frequency bin 1 ... Frequency bin N
    ...
    Beam N
        Frequency bin 1 ... Frequency bin N
Record 2
    Beam 1
        Frequency bin 1 ... Frequency bin N
    Beam 2
        Frequency bin 1 ... Frequency bin N
    ...
    Beam N
        Frequency bin 1 ... Frequency bin N
...
Record N
    Beam 1
        Frequency bin 1 ... Frequency bin N
    Beam 2
        Frequency bin 1 ... Frequency bin N
    ...
    Beam N
        Frequency bin 1 ... Frequency bin N
```

For example, the first four bytes in the file is the noise value for the first frequency bin, in the first beam, of the first record. The last four bytes are for the last frequency bin, of the last beam, of the last record.

E.1 Conversion

The files we have at DRDC Atlantic are not in this format natively. Our files are self-describing (in the IDL environment) and are arranged in little-endian format (Intel architecture native). To use one of these files the self describing header must be removed and the byte order of the data must be reversed.

The header is plain text and can be removed with a hex editor. The byte order can be changed using the Convert class in the Sonar jar file after the header is removed. To run the conversion utility:

```
java -classpath sonar-3.0.0-vmsfom-3.0.0.jar Sonar.Convert inputFile outputFile
```

Where:

Sonar-3.0.0-vmsfom-3.0.0.jar is the Sonar 3 jar file in the installation “lib” directory.

InputFile is the name of the little-endian file to convert.

OutputFile is the desired name of the big-endian output file.

Example: to convert the little-endian file “raw_noise.dat” the command line is:

```
java -classpath sonar-3.0.0-vmsfom-3.0.0.jar Sonar.Convert raw_noise.dat converted.dat
```

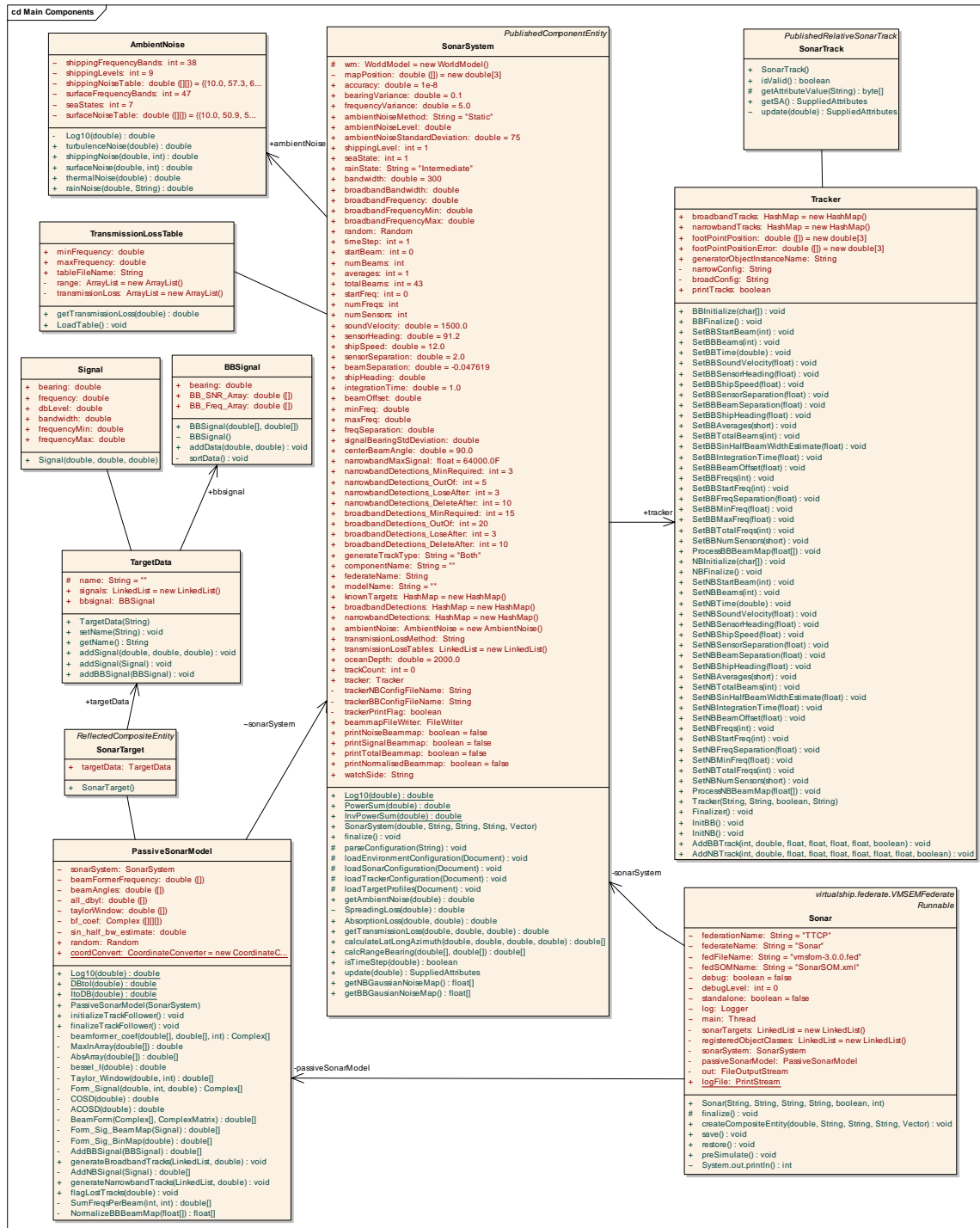
E.2 Usage by Sonar 3

The noise data in these files is not unprocessed pressure or intensity. To create these files the recorded ocean noise has been processed to remove known signals and then normalized. As a result this data cannot be used directly, but instead is used as weightings for the noise values calculated from Wentz curves (AmbientNoiseMethod = Calculate). Without a noise data file the “calculate” method uses a Gaussian distribution to provide fluxuations.

To use a noise file a single “NoiseFile” tag must appear in the Environment section of the Sonar configuration file. Since the files are not self-describing this tag must also specify the number of records, beams, and frequency bins. Please see the section on configuration for more details.

The noise data is used without consideration for the direction of the beams or the real-world frequencies that each frequency bin relates to. The model simply grabs a random starting beam and frequency cell and reads the data required from there. Every subsequent use of the data advances the starting position by one record. The data is self-consistent with regards to beams, frequency cells, and time (record) so the statistics will match the real ocean environment much more closely than a Gaussian distribution does.

Annex F. UML class diagram



List of symbols/abbreviations/acronyms/initialisms

GUI	Graphical User Interface
DLL	Dynamic Link Library
DMSO	Defence Modelling and Simulation Office
DND	Department of National Defence
DRDC	Defence Research and Development Canada
DSTO	Defence Science and Technology Organisation
HLA	High Level Architecture
JAR	Java Archive
R&D	Research & Development
RTI	Run Time Infrastructure
SDK	Software Development Kit
TTCP	The Technical Cooperation Program
VMSA	Virtual Maritime Systems Architecture

Distribution list

Document DRDC Atlantic TM 2005-286:

LIST PART 1: Internal Distribution by Centre:

- 2 (1 CD and 1 Hard copy) DRDC Atlantic Library File Copies
- 3 (CDs) DRDC Atlantic Library (Spares)
- 1 Author

6 TOTAL LIST PART 1

LIST PART 2: External Distribution by DRDKIM

- 1 (CD) NDHQ/DRDKIM
- 1 (CD) Dr. Shane Canney, DSTO Edinburgh, PO Box 1500 Edinburgh, SA 5111, Australia

2 TOTAL LIST PART 2

8 TOTAL COPIES REQUIRED

This page intentionally left blank.

DOCUMENT CONTROL DATA		
(Security classification of title, body of abstract and indexing annotation must be entered when the overall document is classified)		
1. ORIGINATOR (the name and address of the organization preparing the document. Organizations for whom the document was prepared, e.g. Centre sponsoring a contractor's report, or tasking agency, are entered in section 8.)	2. SECURITY CLASSIFICATION (overall security classification of the document including special warning terms if applicable).	
DRDC Atlantic	UNCLASSIFIED	
3. TITLE (the complete document title as indicated on the title page. Its classification should be indicated by the appropriate abbreviation (S,C,R or U) in parentheses after the title).		
Sonar 3 VMSA Federate : user Guide and Technical Description		
4. AUTHORS (Last name, first name, middle initial. If military, show rank, e.g. Doe, Maj. John E.)		
Gillis, Allan D.		
5. DATE OF PUBLICATION (month and year of publication of document)	6a. NO. OF PAGES (total containing information Include Annexes, Appendices, etc).	6b. NO. OF REFS (total cited in document)
April 2007	47 (approx.)	9
7. DESCRIPTIVE NOTES (the category of the document, e.g. technical report, technical note or memorandum. If appropriate, enter the type of report, e.g. interim, progress, summary, annual or final. Give the inclusive dates when a specific reporting period is covered).		
Technical Memorandum		
8. SPONSORING ACTIVITY (the name of the department project office or laboratory sponsoring the research and development. Include address).		
Defence R&D Canada – Atlantic PO Box 1012 Dartmouth, NS, Canada B2Y 3Z7		
9a. PROJECT OR GRANT NO. (if appropriate, the applicable research and development project or grant number under which the document was written. Please specify whether project or grant).	9b. CONTRACT NO. (if appropriate, the applicable number under which the document was written).	
11BK		
10a ORIGINATOR'S DOCUMENT NUMBER (the official document number by which the document is identified by the originating activity. This number must be unique to this document.)	10b OTHER DOCUMENT NOS. (Any other numbers which may be assigned this document either by the originator or by the sponsor.)	
DRDC Atlantic TM 2005-286		
11. DOCUMENT AVAILABILITY (any limitations on further dissemination of the document, other than those imposed by security classification)		
<input checked="" type="checkbox"/> (X) Unlimited distribution <input type="checkbox"/> () Defence departments and defence contractors; further distribution only as approved <input type="checkbox"/> () Defence departments and Canadian defence contractors; further distribution only as approved <input type="checkbox"/> () Government departments and agencies; further distribution only as approved <input type="checkbox"/> () Defence departments; further distribution only as approved <input type="checkbox"/> () Other (please specify):		
12. DOCUMENT ANNOUNCEMENT (any limitation to the bibliographic announcement of this document. This will normally correspond to the Document Availability (11). However, where further distribution (beyond the audience specified in (11) is possible, a wider announcement audience may be selected).		

13. **ABSTRACT** (a brief and factual summary of the document. It may also appear elsewhere in the body of the document itself. It is highly desirable that the abstract of classified documents be unclassified. Each paragraph of the abstract shall begin with an indication of the security classification of the information in the paragraph (unless the document itself is unclassified) represented as (S), (C), (R), or (U). It is not necessary to include here abstracts in both official languages unless the text is bilingual).

This federate provides a medium fidelity sonar model for Virtual Maritime Systems Architecture (VMSA) high level architecture (HLA) simulations. The model best represents towed array systems, but can be configured to model flank arrays as well. The Sonar model is coupled with an auto-detector/signal follower that creates VMSA Sonar tracks without an operator.

While the federate is written in Java, the signal follower is provided only as a compiled Windows DLL. This limits the federate to the Windows operating system, either Windows 2000 or XP.

This document describes the federate software, how to use it, and how to integrate the federate into DRDC Atlantic's VMSA execution system. As well, the software design and technical details are explained.

This document covers version 3.0.1 of the Sonar federate.

14. **KEYWORDS, DESCRIPTORS or IDENTIFIERS** (technically meaningful terms or short phrases that characterize a document and could be helpful in cataloguing the document. They should be selected so that no security classification is required. Identifiers, such as equipment model designation, trade name, military project code name, geographic location may also be included. If possible keywords should be selected from a published thesaurus. e.g. Thesaurus of Engineering and Scientific Terms (TEST) and that thesaurus-identified. If it not possible to select indexing terms which are Unclassified, the classification of each should be indicated as with the title).

Sonar; VMSA; Federate; HLA; Virtual Maritime Systems Architecture; User Guide; Technical Description; Manual

This page intentionally left blank.

Defence R&D Canada

Canada's leader in defence
and National Security
Science and Technology

R & D pour la défense Canada

Chef de file au Canada en matière
de science et de technologie pour
la défense et la sécurité nationale



www.drdc-rddc.gc.ca